CommonsSSLProposal

Abstract

Commons-SSL makes SSL and Java easier.

```
Target:    Jakarta Commons
Sponsor:   Incubator
Champion:  Martin van den Bemt <mvdb@apache.org>
Mentor:    Henri Yandell <bayard@apache.org>
Resources: SVN:     https://svn.apache.org/repos/asf/incubator/commons-ssl/
           Website: http://incubator.apache.org/commons-ssl/
           Jira:    http://issues.apache.org/jira/browse/COMMONS-SSL
           Wiki:    http://wiki.apache.org/commons-ssl/
           Mailing lists:
                    commons-user@jakarta.apache.org
                    commons-dev@jakarta.apache.org
                    commons-cvs@jakarta.apache.org
                    commons-private@jakarta.apache.org ???

Initial committers:
           Julius Davies <juliusdavies@cucbc.com>,
           Oleg Kalnichevski <olegk@apache.org>

Initial source:
           http://juliusdavies.ca/commons-ssl/

Technologies:       Java 1.3, Java 1.4, Java 5, Java 6, JSSE, JCE
Reference:          http://wiki.apache.org/incubator/CommonsSSLProposal
```

Proposal

Commons-SSL gives developers control over many TLS/SSL options, including:

- Client Authentication (using PKCS 12, Java Keystore, or OpenSSL private key formats).
- Managing trust without touching Java's `jre/lib/security/cacerts` file.
- Enable/Disable Hostname Checking (CN, and subjectAlt, with or without wildcards).
- Enable/Disable Certificate Expiry Checking.
- Enable/Disable CRL checking.
- Enable/Disable OCSP checking.
- Enable/Disable Weak Ciphers.
- Enable/Disable Max-Depth checking of certificate chain.

Commons-SSL generates SSL{{`Socket}}`Factory, SSLServerSocketFactory, and SSL`Context objects for consumers to drop into their existing code bases.

Commons-SSL also includes some command-line utilities to help people with common problems:

org.apache.commons.ssl.Ping	*Modeled on `openssl s_client` for debugging SSL issues.*
org.apache.commons.ssl.Key{{`Store}}`Builder	*Utility to convert OpenSSL private keys to Java keystore format, and vice versa.*

Commons-SSL is also able to encrypt/decrypt files in a way that is compatible with the OpenSSL `enc` command.

Please visit http://juliusdavies.ca/commons-ssl/ssl.html for more detailed information.

Background

This java library was developed by Credit Union Central of British Columbia (http://www.cucbc.com/) throughout 2006 with Apache Commons in mind. The original idea was inspired by the EasySSLProtocolSocketFactory and AuthSSLProtocolSocketFactory examples written by Oleg Kalnichevski and Adrian Sutton.

If you google for "java ssl" you'll notice that HTTPClient is actually the first "*.apache.org" site to appear (sometimes 8th page, sometimes 9th page of results):

http://www.google.ca/search?q=java+ssl

Meanwhile if you just google for "ssl", you'll notice that "www.modssl.org" and "www.apache-ssl.org" are on the first page of results. Both of those sites contain versions of the Apache SSL FAQ:

http://httpd.apache.org/docs/2.2/ssl/ssl_faq.html#selfcert

Java developers find these two different "ways" of setting up SSL to be a source of confusion. The OpenSSL/Apache documentation about setting up SSL is very good, very handy. But it just doesn't work with Java, because Java only understands "JKS" files created by "keytool". (Oh wait... months later one usually discovers that Java can also handle "PKCS #12".)

Commons-SSL attempts to make it possible to work in both ways. With Commons-SSL the user no longer needs to know:

1. What kind of keystore file is this? JKS or PKCS #12?

2. The user created a PKCS #12 file using "openssl" but forgot to say "-outform DER".

3. The user created a "Traditional SSLeay" private key instead of PKCS8 by accident.

If Commons-SSL could have a motto, it would be this: "You bring the files and the password, I'll do the rest." Perhaps add this sub-title: "If your password is wrong I'll throw a Probably{{`Bad}}`Password`Exception".

Whereas pre-Commons-SSL, Java's motto would be: "You bring the files, the password, make sure you know the type, and if it's PKCS #12, make sure it's in DER. If your RSA private key and associated X509 chain are in separate files, you're going to have to transform them into PEM, start up Windows Notepad to cut & paste them into a single file, and then use 'openssl pkcs12' to get them into DER."

Perhaps add this sub-title: "If your password is wrong I might throw a Padding{{`Exception or a Digest}}`Exception. Depends on the vendor and version of Java.".

Once the private key is loaded, SSL can start happening. But there are still many options and advanced usages of SSL that a developer might want to use. In Java changing aspects of an SSL{{`Socket}}`Factory or SSLServerSocket`Factory is quite awkward. Many of the approaches in use today end up polluting the entire JVM. You want to connect to "ht``tps://my-self-signed-site.com/" and suddenly all your LDAPS and RMI-SSL and JDBC-SSL calls aren't checking the server certificate either! The HTTPClient group came up with a very novel solution: "https-easy://".

Commons-SSL tried to take this further and generalize it. The developer can now create a single isolated SSL{{`Socket}}`Factory very easily (SSL{{`lient extends SSL}}`Socket`Factory).

```
SSLClient client = new SSLCLient();
```

[Notice how we're borrowing from Http``Client's own usage pattern:
Http{{`Client client = new Http}}`Client();]

The developer can then modify that SSL{{`Socket}}`Factory to suit their needs:

```
SSLClient client = new SSLCLient();

char[] pwd = "secret".toCharArray();

client.setCheckHostname( false );
client.setCheckCRL( false );
client.setCheckExpiry( true );
client.setMaxTrustChainDepth( 6 );  // not yet implemented

client.addTrustMaterial( new TrustMaterial( "/path/to/cert.pem" ) );
client.setKeyMaterial( new KeyMaterial( "/path/to/key.pem", "/path/to/certs.pem", pwd ) );
```

Current Status

Meritocracy

So far it's just been the main developer committing whenever he likes. This has been convenient to try and get the library into a useable state. The library has good coherency partly because only one person has been working on it so far. But on the other hand, SSL is touchy stuff, and it needs eyes looking at it. We will never be able to declare that "yes, this library doesn't compromise your security" without more committers on board.

There is a lot of Java-SSL expertise spread around Apache:

- Tomcat
- Commons-Net
- HttpComponents

- Synapse
- Directory and MINA (Some recent STARTTLS activity!)

These are just five examples that immediately spring to mind, and of course there are many, many more. Hopefully the SSL/TLS experts on some of these projects will take an interest in Commons-SSL.

Community

The community is very new, but they are enthusiastic. A "not-yet-commons-ssl" mailing list was started on January 5th as not-yet-commons-ssl@lists.juliusdavies.ca, and already within 2 months, out of 10 subscribers so far, three are quite active and helpful.

There is a good relationship with the BouncyCastle project.

There is a strong relationship with the HttpComponents team. The main Commons-SSL developer tries to float design ideas by httpcomponents-dev. The recent 0.3.7 release of not-yet-commons-ssl.jar was done primarily to provide support for the new httpcomponents NIO-SSL module.

It's probably a coincidence, but "SSL and Java" related queries on the OpenSSL-User mailing list have completely disappeared since the library was announced on that list. SSL related questions seem to have diminished on the HttpClient-User list as well.

We're currently averaging 150 - 200 downloads a month (from 150 - 200 unique IP addresses).

Core Developers

Definitely a weak spot with Commons-SSL right now. There's only the main developer, with input on the mailing lists coming mostly from HttpComponent's Roland and Oleg. Recently there's been some input on not-yet-commons-ssl@lists.juliusdavies.ca from Mike Ressler and Steve Davidson.

Alignment with existing Apache subprojects

Here's where Commons-SSL is very strong. It's actually allowing all these server-socket-creating Java applications so point to the Apache httpd TLS FAQ!

http://httpd.apache.org/docs/2.2/ssl/ssl_faq.html#selfcert

We're bridging an old split, between OpenSSL and Sun's Java, perhaps like the Pope's recent visit to Turkey?

In terms of code use: we're using code from directory.apache.org, we've also using code from Commons-Codec. In terms of helping other projects: Commons-SSL has lots of hooks in place for HttpComponents and anyone who uses SSL{{`Socket}}`Factory, SSLServerSocketFactory, or SSL`Context can drop Commons-SSL into their project quite seamlessly.

I think Commons-SSL would be good for many open-source projects. Here are a few ideas:

- http://jtds.sourceforge.net/
- http://telnetd.sourceforge.net/
- http://directory.apache.org/ (to help when being an ldaps:// server)
- http://tomcat.apache.org/ (so support APR-SSL config when APR isn't around)

That's just the server side. It's very handy on the client side, too. Already three open source projects (that we know of) are using Commons-SSL:

- http://xfire.codehaus.org/HTTP+Transport
- http://www.soapui.org/
- http://www.bedework.org/

The library recently appeared in Shibboleth's source tree, but we can't find any use of it!

- http://svn.middleware.georgetown.edu/view/trunk/lib/?root=java-xmltooling

Known Risks

Orphaned products

The main developer has been supporting users on the httpclient-user mailing list for 2 years. He's been supporting the Commons-SSL library for 9 months so far, issuing regular releases along the way. Commons-SSL is not going to be orphaned any time soon, whether incubation happens or not!

Inexperience with Open Source

Commons-SSL has been an Open Source project already for 9 months, with mailing-lists for 2 months. The code was derived from HttpClient code in its "contrib" repository. The main developer so far has been active on open source mailing lists for 3 years.

Inexperience with Apache will definitely cause many fumbles on the incubator and commons-dev mailing lists. This will probably frustrate more experienced committers. We'll try our best, and remain tenacious.

Homogeneous Developers

Homogeneity is not really the problem. The problem is lack of developers. But this is common for small "Jakarta-Commons" sub-projects. Nonetheless, Jakarta-Commons is quite a vibrant part of Apache.

Relationships with Other Apache Products

The relationship with HttpComponents is strong, and a relationship with Synapse is just starting. The library seems to be very popular in the web-services world (SoapUI, XFire), and indeed, webservices is exactly where Credit Union Central of British Columbia (the original developer) uses the code inside their shop.

We hope to eventually be the one library everyone in the Java world uses when doing SSL as a server or client, blocking or non-blocking.

The library was written particularly with Tomcat in mind. There is currently no relationship with Tomcat, and more than anything this goal is like a guiding mirage rather than something that actually matters. The Tomcat goal has helped give the library a nice shape. The symmetry with SSL{{`Server and SSL}} `Client is pleasing.

A Excessive Fascination with the Apache Brand

A few people on the not-yet-commons-ssl mailing list have asked when this code will become Apache code. The Apache brand is going to help people feel safer about bringing this SSL security-sensitive code into their shop -
there's no question about that.

Perhaps there is some excessive fascination with the Apache Brand in this case, although we like to think the fascination was more with Apache-Jakarta-Commons than just "Apache". HttpClient is an excellent library! This SSL code seemed like a good way to partially contribute back to the community in kind. Jakarta-Commons seems like a good fit for these 100 java classes. The HttpComponents team was also very encouraging when Commons-SSL was first announced, and that was exciting.

By the way, sorry for the original "common-ssl" name. That was an unfortunate and foolish mistake.

Documentation

http://juliusdavies.ca/commons-ssl/

Initial Source

http://juliusdavies.ca/commons-ssl/download.html

Source and Intellectual Property Submission Plan

- Julius Davies's CLA submitted and accepted.
- http://www.cucbc.com/ Credit Union Central of British Columbia's CCLA and Software Grant submitted and accepted.

External Dependencies

Compile-time dependencies on HttpClient 3.x and Log4J 1.2.x.

No run-time depedencies.

Confession:
We did copy & paste Base64.java from Commons-Codec, and ASN.1 parsing from directory.apache.org But "java -jar common-ssl.jar" wouldn't work without this, and the library is still under 250 KB. We hope to move these out once JSR-277 and Java-7 are ubiquitous (probably in 5 - 7 years).

Cryptography

Yes!